

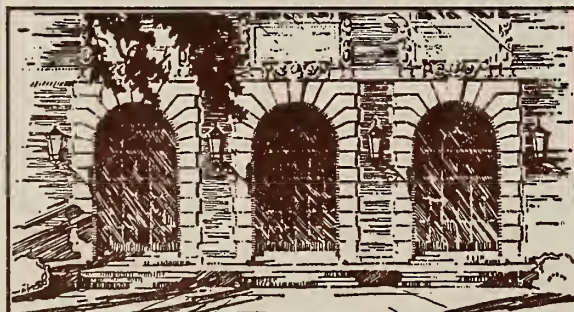
LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84

I26r

no. 571-576

cop. 2



The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

DEC 14 1976
DEC 12 REC'D

SEP 10 1982

LOAN 10 1982

AUG 29 1997

Math

6N

572

p. 2

RATION--A Rational Arithmetic Package

by

R. L. Brown

June 1973

THE LIBRARY OF THE

MAY 23 1973

UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

MAY 31 1973



DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-73-572

RATION--A Rational Arithmetic Package*

by

R. L. Brown

June 1973

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

ABSTRACT

In section A, the theoretical basis for representing a rational number using the prime factorization of its minimal integer ratio is given, and the resulting algorithms for performing addition, subtraction, multiplication, and division are developed. The conjecture that these algorithms will normally require fewer machine operations than algorithms based on storing rational numbers as two long integers is supported by a worst case machine operation count estimate for each. These operation counts are applied to parameters derived from test results.

Section B is a self-contained user's manual for RATION, a rational arithmetic package written in FORTRAN and based on the above algorithms.



Digitized by the Internet Archive
in 2013

<http://archive.org/details/rationalarationalala572brow>

TABLE OF CONTENTS

	<u>Page</u>
A. RATIONAL ARITHMETIC ALGORITHMS.	1
1. <u>Theoretical Foundations</u>	1
2. <u>Conversion Algorithms</u>	4
3. <u>Arithmetic Algorithms</u>	7
4. <u>Package Organization</u>	11
B. RATION--USER'S MANUAL	17
5. <u>User Program</u>	17
6. <u>I/O and Utility Instructions</u>	18
7. <u>Arithmetic Instructions</u>	20
LIST OF REFERENCES.	22
APPENDICES	
I. Sample Program.	23
II. RATION Listing.	25

A. RATIONAL ARITHMETIC ALGORITHMS

1. Theoretical Foundations

A package of FORTRAN subroutines has been programmed to perform arithmetic operations on a special representation for rational numbers using algorithms that attempt to minimize computer execution time at the expense of space required for the storage of the operands and the program itself. The algorithms and representations depend on some elementary properties of the sets of integers and rational numbers as developed in number theory. These properties are defined below. See [1] for fuller explanations.

Definition 1: The set \underline{I} of integers consists of the counting (natural) numbers $(1, 2, 3, \dots)$ and their negatives (additive inverses) and 0.

Definition 2: R is an element of the set \underline{R} of rational numbers if it is any ration I_1/I_2 , $I_1 \in \underline{I}$, $I_2 \in \underline{I}$. Note that this defines an equivalence class for each rational number. The rational number is the equivalence class $(I_3 * I_1)/(I_3 * I_2)$ for $I_3 \in \underline{I}$. If $I_2 > 0$ and there exists no integer I_4 such that I_4 divides both I_1 and I_2 , then I_1/I_2 is the minimal element of the equivalence class and will be referred to as the rational number $R = I_1/I_2$. See [6] for further explanation.

Definition 3: A prime number is any positive integer which is divisible without remainder by exactly two distinct integers--1 and itself.

Definition 4: Any integer I can be uniquely represented by the product of some set of prime numbers. This unique set is called the prime factorization of I .

Using the above concepts, one can see that any rational number involves two integers--each of which has a unique prime factorization. The prime factorization has a unique representation when placed in the form

$$I_1 = \hat{p}_1^{\alpha_1} \hat{p}_2^{\alpha_2} \dots \hat{p}_n^{\alpha_n}$$

$$\alpha_i > 0$$

where \hat{p}_i is the i -th ordered distinct prime appearing in the factorization and α_i is the number of times it appears.

Since $\alpha_i > 0$, a representation for $1/I_2$ is

$$\tilde{p}_1^{-\beta_1} \tilde{p}_2^{-\beta_2} \dots \tilde{p}_m^{-\beta_m}$$

$$\beta_i > 0.$$

Since the rational number R is I_1/I_2 , a representation for R is

$$p_1^{\hat{\alpha}_1 - \hat{\beta}_1} p_2^{\hat{\alpha}_2 - \hat{\beta}_2} \dots p_q^{\hat{\alpha}_q - \hat{\beta}_q}$$

where $\{p_i\}$ is a reordering to include all elements of $\{\tilde{p}_i\}$, $\{\hat{p}_i\}$, and

$$\hat{\alpha}_i = 0 \text{ if } p_i \notin \{\hat{p}_i\}$$

$$\hat{\beta}_i = 0 \text{ if } p_i \notin \{\tilde{p}_i\}.$$

This representation can be used in a computer by storing a string of nodes, each node containing the exponent of a prime associated with the node. If we let an indexed array $P(I)$ contain our primes with $P(1)$ equal to the total number of primes defined in the array (this may be less than its dimension) and $P(2) = 2$, $P(3) = 3$, $P(4) = 5$, $P(5) = 7, \dots$, then each node will take the form of $(\#p_i^{\gamma_i})$ where γ_i is the exponent $\hat{\alpha}_i - \hat{\beta}_i$ of the prime p_i with index $\#p_i$.

One additional node is needed to specify the sign of the rational number and the number of nodes occurring in this particular rational number's representation since this will seldom be the full length of the array. By using a FORTRAN array dimensioned $R(2,K)$, $10 \leq K \leq 25$, we can set $R(1,1)$ so

$$\text{signum}[R(1,1)] = \text{signum}[R]^*$$

$$|R(1,1)| = \text{number of nodes including the bookkeeping node.}$$

Also set $R(2,1) = \text{index of largest prime in the array}$, $R(2,J) = \text{index of prime number taken to the } R(1,J) \text{ power } (J \geq 2)$. In keeping with our notation, $-1, 0, 1$ have $R(1,1) = -1, 0, 1$, respectively, and $R(2,1) = 1$ by convention.

Henceforth, a rational number $R(2,K)$ in this form will be called the indexed prime factorization representation (factored form for short) of the rational number R . It will be written in the form used in the following examples.

$$\text{Ex. 1: } 3/4 = \begin{pmatrix} 3 & -2 & 1 \\ 3 & 2 & 3 \end{pmatrix}$$

$$\text{Ex. 2: } -121/54 = \begin{pmatrix} -4 & -1 & -3 & 2 \\ 5 & 2 & 3 & 5 \end{pmatrix}$$

$$\text{Ex. 3: } 256/41 = \begin{pmatrix} 3 & 7 & -1 \\ 13 & 2 & 13 \end{pmatrix}$$

A representation will also be needed for very long integers if numerators are to be added or subtracted. If a suitable base is chosen such that any number multiplying a digit of the integer will leave the product completely within one computer word, thus retaining the carry, an integer can

$$* \quad \text{signum}[A] = \begin{cases} -1 & \text{if } A < 0 \\ 0 & \text{if } A = 0 \\ 1 & \text{if } A > 0 \end{cases}$$

be represented as a string of computer words, each word representing an (extremely large) digit. Again, a bookkeeping entry is needed to specify the number of digits defined and the sign of the integer I. Otherwise, all entries of the array would have to be zeroed before a change is made. Therefore, $|P(1)|$ is set to the length of the array, $\text{signum}[P(1)]$ to $\text{signum}[I]$, and $P(J)$ to the coefficients of $\text{BASE}^{*(J-2)}$ for $J \geq 2$. The following examples illustrate the representation of long integers. The base is 10000 so the digits 0000 to 9999 are representable.

$$\text{Ex. 4:} \quad 4792643 = (3,2643,479)$$

$$\text{Ex. 5:} \quad -177133 = (-3,7133,17)$$

$$\begin{aligned} \text{Ex. 6:} \quad & -1 = (-2,1) \\ & 0 = (2,0) \\ & 1 = (2,1) \end{aligned}$$

2. Conversion Algorithms

It will be shown in section 3 that during addition, factored forms of two integers must be converted to long integer forms, added, and the resulting long integer converted back to the factored form. Two algorithms are necessary for these conversions.

Consider the case of a factored form integer A (all positive exponents) and a list of prime numbers none of which exceeds one computer word in length. This is reasonable since $\Pi(x)$ --the number of prime numbers less than x --is approximated by $x/\log_e x$ [2] and on any IBM 360 computer this allows the program to allocate 10^8 prime numbers--much more than could ever be stored effectively. Then if long integer Il is set to 1 and each element $Il(2), \dots$ is multiplied by prime $P(J)$ (carrying properly) and each $P(J)$ is multiplied times $Il \alpha_j$ times as J is stepped through all primes with positive α_j in A, Il is converted to a long integer form of A. This is shown in Algorithm C, which, like all algorithms herein, is expressed in Knuth's notation [3].

ALGORITHM C(A,I1): Convert factored form
integer A to long integer I1

C1.[Initialize.] $I1(1) \leftarrow 2, I1(2) \leftarrow 1, M \leftarrow 2.$

C2.[Prime factor loop.] $J \leftarrow 2.$

C3.[Exponent loop.] $K \leftarrow 1.$

C4.[Long integer multiply.] $L \leftarrow 2, I1(L+1) = 0.$

C5.[Prime P times I1.] $IL(L) \leftarrow IL(L)*P(A(2,J)).$
 If $I1(L) > \text{BASE}$, $I1(L) \leftarrow I1(L) - \lfloor I1(L)/\text{BASE} \rfloor * \text{BASE},$
 $I1(L+1) \leftarrow \lfloor I1(L)/\text{BASE} \rfloor.$
 If $L \neq M$, then $L \leftarrow L+1$, go to C5.
 If $I1(L+1) \neq 0$, $M \leftarrow M+1.$

C6.[End loop.] If $K \neq A(1,J)$, $K \leftarrow K+1$, go to C4.

C7.[End loop.] If $J \neq |A(1,1)|$, $J \leftarrow J+1$, go to C3.

C8.[Set bookkeeping node.] $I1(1) \leftarrow M*\text{signum}(A(1,1)).$

END.

Going the other way (converting a long integer to factored form) one must check if each prime divides the integer, and if so, add 1 to the exponent of that prime in the factored form result; if not, try the next prime stopping when $I1 = 1$. This is shown in Algorithm G. It is assumed that a function $\text{IDIV}(I1,P)$ for dividing a long integer I1 by a single word integer P is available. This would be similar to the multiply routine in C3-C6 of Algorithm C.

ALGORITHM G(I2,A): Convert long integer
I1 to factored form A

G1.[Initialize.] $M \leftarrow 2, I \leftarrow 2$.

G2.[Check for I1 = 1.] If $I1(1) = 2$ and $I1(2) = 1$, then
 $A(1,1) \leftarrow I * \text{signum}(I1(1)), A(2,1) \leftarrow M$, END.

G3.[Check remainder on division by a prime.]
 $TEMP = \text{IDIV}(I1, P(M))$. If there is a remainder,
 $M \leftarrow M+1$, go to G2; otherwise, $I1 \leftarrow TEMP$.

G4.[Increment factor count.] If $M \neq A(2,I)$, then
 $I \leftarrow I+1, A(2,I) \leftarrow M, A(1,I) \leftarrow 0$. Set
 $A(1,I) \leftarrow A(1,I) + 1$. Go to G2.

Algorithm G must access all the primes sequentially so the factored form has the primes in order. Algorithm C accesses the primes sequentially but could be written to access them randomly. If the list of primes is needed sequentially, a sequential access device could easily be used to store the list, although the current program package stores them in core. With these conversion algorithms defined the arithmetic algorithms can be completely specified.

The method for choosing the base for long integers can now be defined also. Since any digit may be multiplied by the largest prime in the list, this product must still fit into one signed integer on the computer in use to allow carries. Thus, $\text{BASE} \leq \text{MAX}/P(P(1))$ --where MAX is the largest signed integer representable on the computer in use--and $P(P(1))$ is the largest prime number that will ever occur in the list of primes $P(I)$.

3. Arithmetic Algorithms

Four operations are closed on the set \underline{R} of rational numbers: addition, subtraction, multiplication, and division [4]. All four of these can be implemented easily and effectively with the previously described representations for rational numbers and integers. In fact, multiplication and division become trivial for factored form rationals.

To show this, consider two factored form rationals

$$A = \begin{pmatrix} \alpha_1 & \alpha_2 \dots \alpha_m \\ \#p_1 & \#p_2 \dots \#p_m \end{pmatrix}, \quad B = \begin{pmatrix} \beta_1 & \beta_2 \dots \beta_n \\ \#p_1 & \#p_2 \dots \#p_n \end{pmatrix}.$$

Since their product is the sum of exponents of common factors, it can be found simply by setting a marker IA , IB for each and advancing it through $(\#p_{IA}^{\alpha_{IA}})$, $(\#p_{IB}^{\beta_{IB}})$. At each configuration reached, either $\#p_{IA} > \#p_{IB}$ or $\#p_{IB} > \#p_{IA}$ or $\#p_{IA} = \#p_{IB}$. In each case, the node in the result, $(\#p_{IC}^{\gamma_i})$ which has its own marker IC , takes on the lower indexed prime and the corresponding exponent and increases the corresponding marker by 1. It also increases its own marker by 1. If the prime index on both operands is the same, their exponents are added. Thus, for the configuration above, $\gamma_{IC} = \beta_{IB}$, $\gamma_{IC} = \alpha_{IA}$, or $\gamma_{IC} = \alpha_{IA} + \beta_{IB}$, respectively. This leads to Algorithm M.

ALGORITHM M(A,B,C): $C \leftarrow A*B$

M0.[Set multiply indicator.] $IN \leftarrow 1$.

M1.[Initialize.] $IA \leftarrow 2, IB \leftarrow 2, IC \leftarrow 2,$
 $II \leftarrow \text{MAX}(A(2,1), B(2,1)), I \leftarrow 2$.

M2.[Search for next larger prime.] If $A(2,IA) = B(2,IB) = I$,
 go to M5. If $A(2,IA) = I$, go to M3. If $B(2,IB) = I$, go
 to M4. Otherwise, $I \leftarrow I+1$.

M3.[A factors C.] $C(2,IC) \leftarrow I, C(1,IC) \leftarrow A(1,IA),$
 $IA \leftarrow IA+1, IC \leftarrow IC+1$. Go to M6.

M4.[B factors C.] $C(2,IC) \leftarrow I, C(1,IC) \leftarrow B(1,IB)*IN,$
 $IB \leftarrow IB+1, IC \leftarrow IC+1$, go to M6.

M5.[A and B factor C.] $C(2,IC) \leftarrow I, C(1,IC) \leftarrow A(1,IA) +$
 $B(1,IB)*IN$. $IA \leftarrow IA+1, IB \leftarrow IB+1, IC \leftarrow IC+1$.

M6.[Loop end.] If $I \neq II$, $I \leftarrow I+1$, go to M2; otherwise,
 if $C(1,I) = 0$, $I \leftarrow I-1$, go to M6.

M7.[Bookkeeping node.] $C(1,1) \leftarrow I*\text{signum}(A(1,1)*B(1,1)).$
 $C(2,1) \leftarrow C(2,I)$. END.

For division, $C = A/B$ --the same method of advancing through the nodes of A and B and comparing the indices of the primes holds, except β_{IB} is replaced by $-\beta_{IB}$ since division is accomplished by subtracting exponents of common factors. This leads to Algorithm D, which sets to -1 a marker to be multiplied by β_{IB} .

ALGORITHM D(A,B,C): $C \leftarrow A/B$

D1.[Set divide indicator.] $IN \leftarrow -1$.

D2.Go to M1.

For addition, a least common denominator (l.c.d.) of the two rational operands must be found. This is actually the least common multiple (l.c.m.) of the two denominators. The denominators of the operands can be found easily since they are representable by extracting only the nodes with negative exponents. Their l.c.m. can be found by choosing the more negative of any negative exponents for any primes in either of the two factored form operands. Taking absolute values gives the least common denominator. This result is shown in Algorithm L.

ALGORITHM L(A,B,R): $R \leftarrow$ least common
denominator of A and B

L1.[Initialize.] $IA \leftarrow 2, IB \leftarrow 2, IR \leftarrow 2, I \leftarrow 2,$
 $II \leftarrow \text{MAX}(A(2,1), B(2,1)).$

L2.[Search for next prime.] If $A(1,IA) = B(1,IB) = I,$
go to L5. If $A(1,IA) = I,$ go to L3.
If $B(1,IB) = I,$ go to L4.

L3.[A factors R.] $R(1,IR) \leftarrow -\text{MIN}(A(1,IA), 0).$
 $IA \leftarrow IA+1.$ If $R(1,IR) \neq 0, R(2,IR) \leftarrow I, IR \leftarrow IR+1.$
Go to L6.

L4.[B factors R.] $R(1,IR) \leftarrow -\text{MIN}(B(1,IB), 0).$
 $IB \leftarrow IB+1.$ If $R(1,IR) \neq 0, R(2,IR) \leftarrow I, IR \leftarrow IR+1.$
Go to L6.

L5.[A and B factor R.] $R(1,IR) = -\text{MIN}(A(1,IA), B(1,IB), 0).$
 $IA \leftarrow IA+1, IB \leftarrow IB+1.$ If $R(1,IR) \neq 0, R(2,IR) \leftarrow I, IR \leftarrow IR+1.$

L6.[Bookkeeping node.] If $I \neq II, I \leftarrow I+1,$ go to L2;
otherwise, $R(1,1) \leftarrow IR, R(2,1) \leftarrow R(2,IR).$

If D represents the l.c.d., then for A,B rational numbers $A*D$, $B*D$ are factored form integers. These integers can be converted to long integers by Algorithm C. If each succeeding element of the corresponding integers $IA(K)$, $K = 2(1)|IA(1)|$, $IB(L)$, $L = 2(1)|IB(1)|$ is added with appropriate carries or borrows as in Algorithm I, the numerator can be found. Note that

$$\text{step}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0. \end{cases}$$

Converting the numerator to factored form using Algorithm G and then dividing by D, the denominator, using Algorithm D, gives the result. Since the signs of the operands do not matter once they are attached to the long form integers, subtraction can be carried out by changing the sign of the subtrahend before and after addition. Thus, addition and subtraction are performed as in Algorithms A and S.

ALGORITHM I($I1, I2$): $I1 \leftarrow I1 + I2$

```

I1.[Initialize.]   $IP1 \leftarrow |I1(1)|$ ,  $IP2 \leftarrow |I2(1)|$ ,
                   $I \leftarrow 2$ ,  $II \leftarrow \text{MAX}(IP1, IP2)$ ,  $CNEG \leftarrow 1$ .

I2.[Get first integer to be positive.]
   If  $I1(1)*I2(2) > 0$ , go to I5.

I3.[Check node lengths.]  If  $IP1 = IP2$ , go to I4;
   otherwise, if  $IP1 > IP2$  and  $I1(1) > 0$ , go to
   I7. Otherwise, go to I6.

I4.[Check integer lengths.]  If  $I1(IP1) > I2(IP2)$ 
   and  $I1(1) > 0$ , go to I7; otherwise, go to I6.

I5.[If both  $< 0$ , reverse sign.]  If  $I1(1) > 0$ , go to I7.

I6.[Reverse signs.]   $I1(1) \leftarrow -I1(1)$ ,  $I2(1) \leftarrow -I2(1)$ ,
    $CNEG \leftarrow -1$ .

I7.[Add integers.]   $I1(I+1) \leftarrow 0$ .   $I1(I) \leftarrow$ 
    $I1(I)*\text{signum}(I1(1))*\text{step}(IP1-I) + I2(I)$ 
    $*\text{signum}(I2(1))*\text{step}(IP2-I) + I1(I)$ .  If
    $I1(I) > \text{BASE}$ ,  $I1(I) \leftarrow I1(I) - \text{BASE}$ ,
    $I2(I+1) \leftarrow 1$ .  If  $I1(I) < 0$ ,  $I1(I) \leftarrow I1(I)$ 
    $+ \text{BASE}$ ,  $I1(I+1) \leftarrow -1$ .  If  $I \neq II$ ,  $I \leftarrow I+1$ ,
   go to I7.

I8.[Restore signs.]  If  $I1(I+1) \neq 0$ ,  $I \leftarrow I+1$ .
    $I1(1) \leftarrow I*CNEG$ .

```

ALGORITHM A(A,B,C): $C \leftarrow A+B$

A0.[Initialize.] $BNEG \leftarrow 1$.
 A1.[Find least common denominator.] Go to L1(A,B,R).
 A2.[$A*R, B*R$.] Go to MO(B,R,T). Go to MO(A,R,C).
 A3.[Convert to long integer.] Go to C1(C,I1).
 Go to C1(T,I2).
 A4.[Add integers.] Go to I1(I1,I2).
 A5.[Convert back.] Go to G1(I1,T).
 A6.[Divide numerator by denominator.] Go to D1(T,R,C).
 A7.[Restore B.] $B(1,1) \leftarrow B(1,1)*BNEG$.

ALGORITHM S(A,B,C): $C \leftarrow A-B$

S1.[Initialize.] $BNEG \leftarrow -1, B(1,1) \leftarrow -B(1,1)$.
 S2.[$A + (-B)$.] Go to A1.

4. Package Organization

Since the four arithmetic operations to be implemented all contain two operands and one result, setting each operator up as a FORTRAN subroutine with three arguments (essentially a three address instruction) is an acceptable form. The user is also provided with limited I/O capabilities involving other FORTRAN subroutines as described in section 7. The two conversion routines from section 2 are additional routines called by the arithmetic operators. Since prime numbers are needed, there is a routine

that has some primes defined by a DATA statement and it can generate more primes when required. If a large number of primes were available on a sequential access device, this would be primarily an I/O file routine.

Since it is possible that one of the operands in an arithmetic operation may also be the result, all code is written so that the result is not finalized until all information is gotten from the operands.

An inspection of the subtraction and division algorithms reveals that these need only prepare some indication that the appropriate operation is being done and then use the addition or multiplication algorithm, respectively. In dividing $C = A/B$, if IN is -1 wherever $C(1,IC) = A(1,IA) + B(1,IB)*IN$ is performed, and IN = +1 when multiplication is being performed, then the division routine need only set IN = -1 and call the multiplication routine.

Similarly, when the subtraction routine is called, by setting an indicator BNEG = -1 and changing the sign of B(1,1) for $C = A+B$, subtraction is performed by calling the addition routine. $B(1,1) = B(1,1)*BNEG$ then changes the operand back to its original form on exiting the addition routine.

An alternative way to organize a rational arithmetic package would be to store two long integers for each rational number with tags denoting one as a numerator and the other as a denominator. The ordinary rules for arithmetic involving multiple digit fractions would hold. For example,

$$\frac{A}{B} * \frac{X}{Y} = \frac{L}{M}$$

would be performed by

$$L' = A * X$$

$$M' = B * Y$$

R = greatest common divisor

(g.c.d.) of L', M' found

by Euclid's algorithm [1]

$$L = L' / R$$

$$M = M' / R$$

all performed using routines for arithmetic in which both operands are long integers.

Addition of $\frac{A}{B} + \frac{X}{Y} = \frac{L}{M}$ would be accomplished as follows.

$$L' = A * Y + B * X$$

$$M' = B * Y$$

$$E = \text{g.c.d. of } L', M'$$

$$L = L' / E$$

$$M = M' / E$$

Again, all arithmetic is performed on long integers.

If machine instructions are divided up into sets according to comparable execution times, then it is possible to estimate the worst case execution times of the algorithms needed with the two different representations for rational numbers. Denote the set {multiply, divide} by *, and {add, subtract} by +. A long integer multiply with one integer m_1 digits long and the other m_2 digits long requires $m_1 * m_2$ *'s and, at worst, $m_1 * m_2$ +'s. Using this, consider the rational numbers $\frac{A}{B}$ with A being n_1 digits long, B being d_1 digits long, and $\frac{X}{Y}$ with X, Y having n_2 , d_2 digits, respectively. Then rational multiplication and division require, at worst, the number of operations indicated in Table 4.1. The value K is the number of divisions

required to find the greatest common divisor R of L' , M' . Lamé [1] has put the upper bound on K as $(\text{BASE}/2)$ while Knuth [3] estimates K as $(12 \log 2/\pi^2) \log N$, for N the minimum of L' , M' . It should be noted that the product of two integers will have, at worst, as many digits as the sum of the number of digits in the operands.

Multiplication

*	+	Reason
$n_1 * n_2$	$n_1 * n_2$	$L' = A * X$
$d_1 * d_2$	$d_1 * d_2$	$M' = B * Y$
$K(n_1 + n_2)(d_1 + d_2)$	$K(n_1 + n_2)(d_1 + d_2)$	$R = \text{g.c.d. of } L', M'$
$(n_1 + n_2) * \min(n_1 + n_2, d_1 + d_2)$	$(n_1 + n_2) * \min(n_1 + n_2, d_1 + d_2)$	$L = L' / R$
$(d_1 + d_2) * \min(n_1 + n_2, d_1 + d_2)$	$(d_1 + d_2) * \min(n_1 + n_2, d_1 + d_2)$	$M = M' / R$

Addition

	$3/2 * \max(n_1 + d_2, n_2 + d_1) +$	
$n_1 * d_2 + n_2 * d_1$	$n_1 * d_2 + n_2 * d_2$	$L' = A * X + B * Y$
$d_1 * d_2$	$d_1 * d_2$	$M' = B * Y$
$K * \max(n_1 + d_2, n_2 + d_1)$	$K * \max(n_1 + d_2, n_2 + d_1)$	$E = \text{g.c.d. of } L', M'$
$*(d_1 + d_2)$	$*(d_1 + d_2)$	
$\min(\max(n_1 + d_2, n_2 + d_1), d_1 + d_2)$	$\min(\max(n_1 + d_2, n_2 + d_1), d_1 + d_2)$	$L = L' / E$
$* \max(n_1 + d_2, n_2 + d_1)$	$* \max(n_1 + d_2, n_2 + d_1)$	
$\min(\max(n_1 + d_2, n_2 + d_1), d_1 + d_2)$	$\min(\max(n_1 + d_2, n_2 + d_1), d_1 + d_2)$	$M = M' / E$
$*(d_1 + d_2)$	$*(d_1 + d_2)$	

Table 4.1 Worst case operation count estimate for integer rational representation algorithms.

In the indexed prime factorization representation, the only multiplications required are due to conversion between factored form and long integers. Table 4.2 shows the worst case operation count estimate for two rational operands A, p nodes long; and B, q nodes long. The value K_1 is the average exponent of the primes, K_{21} is the number of digits in the converted integer I1, K_{22} is that for integer I2, and K_3 is the index of the largest prime in the final result.

In the sample program in Appendix I (using base 1000 integers), average values for the above parameters were obtained. These appear in Table 4.3.

Multiplication

*	+	Reason
	$p+q$	$C(1,I)=A(1,I)+B(1,I)$

Addition

	$p+q$	find g.c.d. R
	$p+q+2\max(p,q)$	$A*R, B*R$
$1/2K_1*(p*K_{21}+q*K_{22})$	$1/4K_1*(p*K_{21}+q*K_{21})$	Convert $A*R \rightarrow I1, B*R \rightarrow I2$
	$3/2 \max(K_{21}, K_{22})$	add integers
$1/2K_3*\max(K_{21}, K_{22})$	$1/4K_3*\max(K_{21}, K_{22})$	convert to factored form result.

Table 4.2. Worst case operation count estimate for
factored form representation algorithms.

PARAMETER	VALUE
n_1	2
n_2	2
d_1	2
d_2	2
K_1	2
K_{21}	2
K_{22}	2
P	3
q	3
K_3	100

Table 4.3. Average observed parameter
values from test program

A considerable savings in execution time for the factored form algorithms over the integer ration algorithms can be predicted on the basis of this test. Table 4.4 lists the comparative average values computed using these parameters. Knuth's estimate for K , based on the average integer being 10^6 , is approximately 12.

MACHINE OPERATIONS	INTEGER RATIO		FACTORED FORM	
	*	+	*	+
*	232	236	0	112
+	232	242	6	77

Table 4.4. Computed comparisons using
test result parameters

5. User Program

The RATION package currently available in the PUBLIC Library and listed in Appendix II was designed to minimize user responsibility for manipulation of rational variables. The user's primary responsibilities in the FORTRAN calling program are to dimension the variables properly and to direct the sequence of I/O and arithmetic operations through FORTRAN subroutine calls. Several basic I/O facilities and all arithmetic operations are provided and diagnostics are available at two levels. The list of prime numbers is generated by the package automatically although all primes less than 1000 are placed in the array P at compilation time. If subroutine RATION is called, a second level of diagnostics can also be invoked. Otherwise, any routine can be called with no further preliminary calls.

All rational variables used by the routine must be specified as A is below:

$$\text{INTEGER*2 } A(2,K)$$

where K is at least 10. Rational arrays can be created simply by adding the array dimensions after the two required dimensions. For example,

$$B(2,10,5,3)$$

is a (5,3) array of rational numbers. When these array elements are used as a subroutine calling argument, the (1,1) element is placed in the argument list. For example, CALL ENTER(B(1,1,2,4)) sets the (2,4) element of B equal to an integer read from cards. The sign of any rational variable is the sign of its (1,1) element; if $B(1,1) < 0$, B is negative.

The program can be initialized by

$$\text{CALL RATION}(N)$$

where N is the value of K if all rational variables are dimensioned (2,K). If during execution any rational variable requires K larger than N, a message to that effect is printed. If N is set to 0, no checking is done.

Any integers that are not factorable in terms of the available primes are placed at the end of the prime array by the conversion routine PTOG. If these integers (to the base PBASE) are more than one word long, all succeeding words of it are tagged with a minus sign. When converting back to integer representations, PI is initialized to this unfactorable integer instead of to 1. In this way, rational numbers of reasonable complexity can be computed.

Default diagnostics are printed when an attempt is made to divide by zero. The quotient is set to zero and the program continues. If a conversion routine GTOP (see section 6) is called to convert a factored non-integer to integer form, a message is printed.

6. I/O and Utility Instructions

Two input routines are provided for creating factored form integers. These can then be divided to create rational variable values. One output routine writes a fixed format rational as a signed two integer quotient. User calls for the subroutines are listed below.

CALL ENTER(A): reads an integer from FORTRAN FILE 5 (card reader). The format of the required cards follows. The first card contains the least integer greater than or equal to the number of decimal digits divided by four, right-justified in columns 1-5; -1, +1 in columns 6, 7 indicate the integer's sign. The base 10 digits are then divided in groups of four from the least significant digit up. These groups are placed starting with the most significant group

in each four columns starting with column 1.

For example, -177133 has input

	#00002-1#				
	#00177133				#
column	1	5	8		80

CALL CREATE(I,A): can be used if the integer I can be represented in an INTEGER*4 computer word. I is a signed integer which becomes the value of A on exiting the subroutine.

CALL WRITE(A): outputs A in the form

$$\pm 1 * - \frac{I_1}{I_2} - -$$

Any additional comments or spacing can be provided by FORTRAN I/O statements in the user program.

If the user feels he understands the representations for rational numbers and integers as presented in section 1, he can also use two utility routines that are called by the I/O and arithmetic routines. These provide the conversions described in section 2. If I is a long integer variable and A is a factored form rational variable, then CALL PTOG(I,A) transforms I to A and CALL GTOG(A,I) transforms A to I. Thus, the program in Figure 6.1. places the factored form of 177133 into A. The user can save execution time by preparing constants in this and similar ways, but only if he understands what he is doing.

```

INTEGER*2 A(2,10)

INTEGER*4 I(3)

DATA I/3, 7133, 17/

CALL GTOP(I,A)

```

Figure 6.1. Sample initialization of
factored form integer

7. Arithmetic Instructions

Four arithmetic routines are provided--one for each of the four operations closed on the rational numbers. RADD and RMPY are the principle routines; RSUB and RDIV are supplementary entries to these using the 360 FORTRAN option of multiple entries [5]. In compilers without this feature, RSUB and RDIV could be very short routines that call RADD and RMPY, respectively, after setting a special variable in a COMMON array. The routines are called as follows for A, B, C factored form rational variables.

```

CALL RMPY(A,B,C):  C = A*B
CALL RDIV(A,B,C):  C = A/B
CALL RADD(A,B,C):  C = A + B
CALL RSUB(A,B,C):  C = A - B

```

Any operators can be redundant.

```
CALL RADD(A,A,A)
```

will double A, but much more expensively than

```
CALL RMPY(TWO,A,A)
```

for TWO the factored form of two.

Logical comparisons may be made using the fact that the sign of a rational variable is the sign of its (1,1) element. To compare A with B,

first compare the sign of A(1,1) with that of B(1,1). If this does not clarify matters, perform

```
CALL RSUB(A,B,TEMP)
```

and compare the sign of TEMP(1,1). Obviously, this is quite time-consuming since RSUB is the most expensive routine to execute; however, comparisons are possible.

A sample program using all arithmetic routines and the I/O routines other than ENTER is in Appendix I.

LIST OF REFERENCES

- [1] Ore, O., NUMBER THEORY AND ITS HISTORY, McGraw-Hill: New York (1948).
- [2] Niven, I. and Zuckerman, H. S., AN INTRODUCTION TO THE THEORY OF NUMBERS, John Wiley & Sons, Inc.: New York (1972).
- [3] Knuth, D. E., THE ART OF COMPUTER PROGRAMMING, vol. 1, Addison-Wesley: Massachusetts (1968).
- [4] Peterson, J. A. and Hashisaki, J., THEORY OF ARITHMETIC, John Wiley & Sons, Inc.: New York (1967).
- [5] IBM System/360 FORTRAN IV Language, S360-50, New York (1971).
- [6] Wolf, F. L., NUMBER SYSTEMS AND THEIR USES, Xerox College Publishing: Massachusetts (1971).

APPENDIX I

Sample Program


```

IMPLICIT INTEGER*2(A-H,J-O,U-Z)
IMPLICIT INTEGER*4(I,P)
COMMON /COM/ PLIM,PBASE,IPW(100),P(15000)
DIMENSION C(2,25,78),ONE(2,25),TMP(2,25),
1 TWO(2,25)
C
C*****
C
C      THIS PROGRAM COMPUTES THE ELEMENTS IN A LOWER TRIANGULAR
C      12*12 MATRIX.  C(K,M) IS REPRESENTED AS C(K*(K-1)/2+M).
C      THIS PREVENTS STORAGE OF SPARSE ELEMENTS.
C
C*****
C
C      CALL RATION(25)
C
C      C(2,2)=.5
C
C      CALL CREATE(1,ONE)
C      CALL CREATE(2,TWO)
C      CALL RDIV(ONE,TWO,C(1,1,3))
C
C      C(K,2)=C(K-1,2) + 1/(2*(K-1))
C
C      DO 10 I=3,12
C      CALL CREATE(I-1,TMP)
C      CALL RMPY(TWO,TMP,TMP)
C      CALL RDIV(ONE,TMP,TMP)
C      CALL RADD(C(1,1,(I-1)*(I-2)/2+2),TMP,C(1,1,I*(I-1)/2+2))
C      CALL WRITE(C(1,1,I*(I-1)/2+2))
10 CONTINUE
C      DO 20 I=3,12
20 C(1,1,I*(I-1)/2+1)=0
C
C      C(K,M) = C(K-1,M) + (M-1)/(M*(K-1))*C(K-1,M-1)
C      M=3,...,K
C      K=3,...,12
C
C      DO 30 IK=3,12
C      DO 30 IM=3,IK
C      CALL CREATE(IM*(IK-1),TMP)
C      CALL CREATE(IM-1,ONE)
C      CALL RDIV(ONE,TMP,TMP)
C      CALL RMPY(C(1,1,(IK-1)*(IK-2)/2+IM-1),TMP,TMP)
C      CALL RADD(C(1,1,(IK-1)*(IK-2)/2+IM),TMP,C(1,1,IK*(IK-1)/2+IM))
C      CALL WRITE(C(1,1,IK*(IK-1)/2+IM))
C      IF(P(1).GE.17000)GO TO 51
30 CONTINUE
C
C      C(K,1) = 1 + SUM (-1)**I * C(K,I)
C      I=1,...,K
C
C      DO 50 IK=2,12
C      KLM=IK*(IK-1)/2+1
C      CALL CREATE(1,C(1,1,KLM))
C      DO 40 IJ=2,IK
C      C(1,1,IK*(IK-1)/2+IJ)=C(1,1,IK*(IK-1)/2+IJ)*(-1)**(IJ+1)
C      CALL RADD(C(1,1,IK*(IK-1)/2+IJ),C(1,1,KLM),
1 C(1,1,KLM))
40 CONTINUE
C      CALL WRITE(C(1,1,KLM))
50 CONTINUE
51 STOP
END

```


APPENDIX II
RATION Listing


```

C*****
C*****
C
C
C          * * * RATION * * *
C
C*****
C          A PROGRAM BY
C            R.L. BROWN
C            MARCH 6, 1973
C*****
C
C
C THIS PACKAGE READS AND CREATES INTEGERS, FORMS RATIONAL NUMBERS, ADDS
C AND SUBTRACTS, MULTIPLIES AND DIVIDES RATIONAL NUMBERS, AND OUTPUTS
C RATIONAL NUMBERS. ALL RATIONAL VARIABLES ARE INTEGER*2 AND SHOULD
C BE DIMENSIONED C(2,K), K>10 BUT USUALLY LESS THAN 25. INTERNAL
C REGISTERS ARE PLACED IN COMMON BLOCKS COM, RT, P12, TO SERVE SUCH
C PURPOSES AS STORING INTEGER*4 PRIME NUMBERS AND HOLDING
C INTEGER REPRESENTATIONS IN FINDING THE NUMERATOR DURING RATIONAL
C ADDITION. THE SUBROUTINES AVAILABLE ARE:
C   ENTER(A): PLACES AN INTEGER INPUT FROM CARDS INTO RATIONAL
C             VARIABLE A. SEE SUBROUTINE WRITEUP FOR CARD FORMAT.
C
C   CREATE(I,A): PLACES ANY INTEGER OF LESS THAN 16 DECIMAL
C                DIGITS INTO RATIONAL VARIABLE A.
C
C   WRITE(A): WRITES THE CURRENT VALUE OF THE RATIONAL VARIABLE A.
C
C   RMPY(A,B,C): C=A*B          (A,B,C ARE RATIONAL VARIABLES)
C
C   RDIV(A,B,C): C=A/B          ''
C
C   RADD(A,B,C): C=A+B          ''
C
C   RSUB(A,B,C): C=A-B          ''
C
C (GTOP(A,P1), PTGG(P1,A), GENERP ARE CALLABLE BY THE USER IF HE DESIRES
C
C   SEE SUBROUTINE WRITEUPS FOR DETAILS.)
C*****

```

SUBROUTINE RATION(I)

```

C*****
C
C A CALL TO RATION WITH I SET TO SOME POSITIVE INTEGER INITIALIZES
C A CHECKING PROCEDURE WHICH WARNS IF ANY FACTORED FORM RATIONAL
C NUMBER EXCEEDS I IN LENGTH. OTHERWISE, NO SUCH CHECKING IS
C DONE.
C
C*****
C   IMPLICIT INTEGER*2(A-H,J-O,U-Z)
C   IMPLICIT INTEGER*4(I,P)
C   COMMON /COM/PLIM,PBASE,PT(100),P(10000)
C   IF(I)1,1,2
C 1 PLIM=2**30-1
C   GO TO 3
C 2 PLIM=1
C 3 CONTINUE
C   RETURN
C   END

```



```

      SUBROUTINE GTOPI(A,PI)
      IMPLICIT INTEGER*2(A-H,J-Q,Z)
      IMPLICIT INTEGER*4(I,P)
      COMMON /COM/ PLIM,PBASE,PT(100),P(10000)
      DIMENSION A(2,1),PI(1)
C*****
C
C
C  THIS ROUTINE PLACES INDEXED PRIME FACTORIZATION FORM INTEGER A INTO
C  THE INTEGER ARRAY PI.  PI IS INITIALIZED TO 1 AND MULTIPLIED
C  SUCCESSIVELY BY PRIMES TO GET A.  IF A IS NOT COMPLETELY SPECIFIED
C  BY THE AVAILABLE PRIMES, THE UNFACTORED PART IS TAKEN FROM THE
C  END OF THE PRIME ARRAY AND INITIALIZES PI.
C
C
C*****
C      IM IS THE LARGEST DEFINED ELEMENT OF PI
C  INITIALIZE PI TO 1, INITIALIZE CONTROL PARAMETERS.
      IM = 2
      PI(2) = 1
      II = A(1,1)
      II = IABS(II)
      IF(II.LE.1) GO TO 12
      PI(1) = A(1,1)/II
      IF(A(2,1).LE.8000)GO TO 3
      II=II-1
1  PI(IM)=IABS(P(A(2,1)+IM-2))
      IF(P(A(2,1)+IM-1).GE.0 .OR. P(A(2,1)+IM-1).LT.-99999)GO TO 2
      IM=IM+1
      GO TO 1
2  IF(II.EQ.1)GO TO 11
C  MULTIPLY MULTIWORD INTEGER PI BY SINGLE WORD INTEGER P(A(2,1))*IKK
3  DO 10 I=2,11
      IF(A(2,I).GT.P(1).AND.P(1).LT.8000)CALL GENERP
      IF(A(2,I).GT.8000)GO TO 11
      IKK=A(1,I)
      IF(IKK.LT.0)GO TO 13
      DO 10 IK=1,IKK
          IMM = IM
          PD=0
          DO 9 IJ=2,IM
              PT(IJ)=P(A(2,I))*PI(IJ)
              IF(IM.EQ.2)PD=PT(2)/PBASE
              IF(IJ.EQ.2)GO TO 5
              PD=PT(IJ-1)/PBASE
              PT(IJ-1)=PT(IJ-1)-PD*PBASE
              PT(IJ) = PT(IJ)+PD
              PD=PT(IJ)/PBASE
5  IF(IJ.NE.IM.OR.PD.EQ.0)GO TO 3
              PT(IM)=PT(IM)-PD*PBASE
4  IMM=IMM+1
              PT(IMM)=PD
              PD=PT(IMM)/PBASE
              IF(PD.EQ.0) GO TO 7
              PT(IMM)=PT(IMM)-PD*PBASE
              GO TO 4
5  IF(IJ.NE.IM)GO TO 9
7  DO 8 ID=2,IMM
8  PI(ID)=PT(ID)
9  CONTINUE
      IM=IMM
10 CONTINUE
11 PI(1) = PI(1)*IM
      RETURN
C  IF A=<-1,0,+1> THIS SECTION FIXES PI TO THE PROPER FORM FOR THESE
C  SPECIAL VALUES.
12 PI(1) = 2
      PI(2) = II
      IF(II.NE.0) PI(1) = PI(1)*A(1,1)
      RETURN
C  ERROR MESSAGE IF A IS NOT AN INTEGER (POWER OF A PRIME < 0).
13 WRITE(6,999)
999 FORMAT(5X,'GTOP CALLED WITH NON-INTEGGER ARGUMENT')
      RETURN
      END

```



```

      SUBROUTINE PTGG(PI,A)
      IMPLICIT INTEGER*2(A-H,J-O,U-Z)
      IMPLICIT INTEGER*4(I,P)
      COMMON /COM/ PLIM,PBASE,PT(100),P(10000)
      DIMENSION PI(1),A(2,1)
      DATA IL/1/
C*****
C
C
C  THIS ROUTINE CHANGES  INTEGER PI TO ITS INDEXED PRIME REPRESENTATION
C  FORM BY DIVIDING BY SUCCESSIVE PRIMES AND RECORDING THE FACTORS.
C  PI IS REDUCED TO 1 BY THE SUCCESSIVE DIVISIONS.
C  IF THE AVAILABLE PRIMES DON'T COMPLETELY FACTOR PI, THE UNFACTORED
C  PART IS APPENDED TO THE ARRAY OF PRIMES.
C
C*****
C
C  SAVE INTEGER BASE AND INITIALIZE PARAMETERS.
C
      II = IABS(PI(1))
      IF(PI(2).LE.1.AND.II.EQ.2) GO TO 45
      DO 2 I=1,II
      2 PT(I) = 0
      A(1,1) = PI(1)/II
      A(1,2)=0
      A(2,2)=0
      IA=2
      IM = 2
      1 PD=PI(II)
      OUT = 0
      PT(1) = PI(1)
C
C  DIVIDE BY P(IM) AND SEE IF IT'S A FACTOR
C
      DO 10 I=2,II
      IJ = II-I+2
      PE = PD/P(IM)
      PT(IJ) = PE
      IF(IJ.EQ.IABS(PT(1)).AND.PE.EQ.0)
      1 PT(1)=ISIGN(IABS(PT(1))-1,PT(1))
      PD = (PD-PE*P(IM))*PBASE
C
C  SET OUT TO 1 IF PI(IM) IS NOT A FACTOR
C
      IF(IJ.EQ.2.AND.PD.NE.0) OUT = 1
      PD = PD + PT(IJ-1)
      10 CONTINUE
      IF (OUT.EQ.1) GO TO 30
C
C  P(IM) WAS A FACTOR;ADD IT TO A
C
      IF(IM.EQ.A(2,IA).OR.A(2,IA).EQ.0)GO TO 15
      IA=IA+1
      IF(IA.GT.PLIM)GO TO 41
      A(1,IA)=0
      15 A(1,IA)=A(1,IA)+1
      A(2,IA)=IM
      II = IABS(PT(1))
      DO 20 IJ=1,II
      20 PT(IJ) = PT(IJ)
      IF(II.EQ.2.AND.PI(2).EQ.1) GO TO 40
      GO TO 1
      30 IF(IM+1.GT.P(1).AND.P(1).LT.8000) CALL GENERP
      IM=IM+1
      IF(IM.LE.8000)GO TO 1
      IF(A(2,IA).NE.0)IA=IA+1
      A(2,IA)=8000+IL
      A(1,IA)=1
      DO 35 I=2,II
      35 P(8000+I-2+IL)=PI(1)*ISIGN(1,5-2*I)

```



```

      IL=IL+11-1
      P(IL+8000)=-9999999
      GO TO 40
41 WRITE(6,999)
999 FORMAT(5X,'FACTORED FORM VARIABLE NOT LARGE ENOUGH')
40 A(1,1)=IA*A(1,1)
   A(2,1)=A(2,IA)
42 RETURN
C
C IF PI = <-1,0,+1>, SET A TO ITS SPECIAL FORM.
C
45 A(1,1) = PI(2)*PI(1)/11
   A(2,1) = 1
   RETURN
   END

      SUBROUTINE GENERP
      IMPLICIT INTEGER*2(A-L,N-Z)
      REAL*8 PR
      COMMON /CGM/ MLIM,MBASE,MT(100),MP(10000)
C
C*****
C
C THIS ROUTINE GENERATES THE NEXT PRIME IN THE ARRAY MP(I) USING
C KNUTH'S ALGORITHM P, SECTION 1.3.2
C MP(M+1) CONTAINS A POSSIBLE PRIME NUMBER.
C MPQ IS THE QUOTIENT OF DIVIDING MP(M+1) BY A KNOWN PRIME;
C PR IS THE REMAINDER.
C
C
C*****
      M = MP(1)
      MP(M+1) = MP(M) + 2
1   DO 10 I=2,M
      MPQ = MP(M+1)/MP(I)
      IF (MOD(MP(M+1),MP(I)).EQ.0) GO TO 12
      IF (MPQ.LE.MP(I)) GO TO 11
10  CONTINUE
11  MP(1) = M+1
      RETURN
12  MP(M+1)=MP(M+1)+2
      GO TO 1
      END

      BLOCK DATA
      COMMON /CGM/ MLIM,MBASE,MT(100),MP(10000)
      INTEGER*4 MBASE/10000/,MLIM/9999999/
      INTEGER*4 MP/169,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,
+53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,
+137,139,149,151,157,163,167,173,179,181,191,193,197,199,211,
+223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,
+307,311,313,317,331,337,347,349,353,359,367,373,379,383,
+389,397,401,409,419,421,431,433,439,443,449,457,461,463,467,
+479,487,491,499,503,509,521,523,541,547,557,563,569,571,
+577,587,593,599,601,607,613,617,619,631,641,643,647,653,
+659,661,673,677,683,691,701,709,719,727,733,739,743,751,
+757,761,769,773,787,797,809,811,821,823,827,829,839,853,
+857,859,863,877,881,883,887,907,911,919,929,937,941,947,
+953,967,971,977,983,991,997,9831*0/
      END

```



```

SUBROUTINE RMPY(A,B,C)
IMPLICIT INTEGER*2(A-H,J-Q,Z)
IMPLICIT INTEGER*4(I,P)
INTEGER*4 MINO,MAXO
COMMON /CGM/ PLIM,PBASE,PI(100),P(10000)
DIMENSION C(2,1),A(2,1),B(2,1),D(2,50)

```

```

C
C*****
C THIS ROUTINE MULTIPLIES OR DIVIDES BY ADDING OR SUBTRACTING
C THE EXPONENTS OF THE PRIME NUMBERS IN THE FACTORIZATION OF THE
C OPERANDS. IF A PRIME IS DEFINED FOR A BUT NOT FOR B, ITS EXPONENT
C IN B IS 0. ENTRIES ARE:
C RMPY: A*B
C RDIV: A/B
C*****
C INITIALIZE PARAMETERS
C
  IN = 1
  1 PA=A(1,1)
  PB = B(1,1)
  PA=IABS(PA)
  PB=IABS(PB)
  IF(PA*PB.LE.1) GO TO 25
  ID=2
  IA=A(2,1)
  IB=B(2,1)
  II=MAXO(IA,IB)
  IA=MINO(PA,2)
  IB=MINO(PB,2)
  D(2,1)=1
  DO 10 I=2,II
    IF(A(2,IA)*1.EQ.1)GO TO 3
    GO TO 4
  3 IF(A(2,IA).EQ.B(2,IB))GO TO 5
    D(1,ID)=A(1,IA)
    D(2,ID)=A(2,IA)
    IA=MINO(IA+1,PA)
    GO TO 9
  4 IF(B(2,IB)*1.NE.1)GO TO 10
    D(1,ID)=B(1,IB)*IN
    D(2,ID)=B(2,IB)
    IB=MINO(IB+1,PB)
    GO TO 9
  6 D(1,ID)=A(1,IA)+IN*B(1,IB)
    D(2,ID)=B(2,IB)
    IA=MINO(IA+1,PA)
    IB=MINO(IB+1,PB)
  9 IF(I.EQ.II)GO TO 10
    IF(D(1,ID).NE.0) ID=ID+1
    IF(ID.GT.PLIM)GO TO 13
  10 CONTINUE
C D IS TEMPORARY STORAGE SINCE A OR B MAY EQUAL C IN THE CALLING
C ROUTINE.
  IF(D(1,ID).EQ.0)ID=ID-1
  D(1,1)=ISIGN(ID,A(1,1)*B(1,1)*1)
  D(2,1)=D(2,ID)
  GO TO 14
  13 WRITE(6,999)
999 FORMAT(5X,'FACTORED FORM VARIABLE NOT LARGE ENOUGH')
  14 DO 15 I=1,ID
    C(1,I)=D(1,I)
  15 C(2,I)=D(2,I)
  RETURN
C
C IF BOTH A AND B ARE IN <-1,0,+1>, COMPUTE THE SPECIAL FORM OF C.
C IF B IS A ZERO DIVISOR, PRINT AN ERROR MESSAGE.
C
  25 C(1,1) = A(1,1)*B(1,1)
  C(2,1)=1
  IF(PB.EQ.0 .AND. IN.EQ.-1) WRITE(6,1000)
1000 FORMAT(5X,'DIVISION BY ZERO; QUOTIENT SET TO ZERO')
  RETURN
  ENTRY RDIV(A,B,C)
  IN = -1
  GO TO 1
  END

```



```

SUBROUTINE RADD(A,B,C)
IMPLICIT INTEGER*2(A-H,J-Q,Z)
IMPLICIT INTEGER*4(I,P)
INTEGER*4 MINO,MAXO
COMMON /COM/ PLIM,PBASE,PT(100),P(10000)
COMMON /RT/R(2,50),T(2,50)
DIMENSION A(2,1),B(2,1),C(2,1)
COMMON /P12/ P1(50),P2(50)

```

C
C

C*****

C

C

C THIS ROUTINE ADDS OR SUBTRACTS A AND B, PLACING THE RESULTS INTO C.

C ENTRIES ARE:

C

 RADD(A,B,C): A+B=C

C

 RSUB(A,B,C): A-B=C

C

C

C*****

C

C PF(I,P) IS USED IN ADDING TWO INTEGERS OF DIFFERENT LENGTH

C BY ZEROING THE SHORTER WHEN NECESSARY.

C

 PF(IZ,PC)=MAXO(PC-IZ+1,0)/MAXO(PC-IZ+1,1)

 BNEG = 1

 GO TO 1

 ENTRY RSUB(A,B,C)

 BNEG = -1

 B(1,1) = -B(1,1)

1 PA = A(1,1)

 PB = B(1,1)

 PA=IABS(PA)

 PB=IABS(PB)

 IF((A(2,PA-1).GT.8000.AND.PA.NE.2).OR.(B(2,PB-1).GT.8000

1 .AND.PB.NE.2))GO TO 110

 CNEG = 1

 DO 2 I=1,100

 R(1,I)=0

2 PT(I) = 0

 R(1,2)=0

 R(2,1)=1

 IR=2

 II=MAXO(A(2,1)*1,B(2,1)*1)

 IA=MINO(2,PA)

 IB=MINO(2,PB)

 IF(II.LE.1)GO TO 13

C

C FIND THE LEAST COMMON DENOMINATOR BY STORING THE MOST NEGATIVE

C OF ALL NEGATIVE EXPONENTS OF PRIMES.

C

 DO 10 I=2,II

 IF(A(2,IA).EQ.1)GO TO 3

 GO TO 4

3 IF(A(2,IA).EQ.B(2,IB))GO TO 6

 IF(A(1,IA).GE.0)GO TO 5

 R(1,IR)=A(1,IA)

 R(2,IR)=A(2,IA)

 IR=MINO(IR+1,II)

5 IA=MINO(IA+1,PA)

 GO TO 10

4 IF(B(2,IB).NE.1)GO TO 10

 IF(B(1,IB).GE.0) GO TO 7

 R(1,IR)=B(1,IB)

 R(2,IR)=B(2,IB)

 IR=MINO(IR+1,II)

7 IB=MINO(IB+1,PB)

 GO TO 10

6 R(1,IR)=MINO(1*A(1,IA),1*B(1,IB))

 IF(R(1,IR).GE.0)GO TO 9

 R(2,IR)=A(2,IA)

 IR=MINO(IR+1,II)

9 IA=MINO(IA+1,PA)

 IB=MINO(IB+1,PB)

10 CONTINUE


```

      IF(R(1,IR).GE.0)IR=IR-1
      R(1,1)=IR
      R(2,1)=R(2,IR)
      GO TO 14
13  R(1,1)=1
      R(2,1)=1
C
C  PUT PARTIAL NUMERATORS INTO T AND C BY A*R, B*R, WHERE R IS THE
C  LEAST COMMON DENOMINATOR.  CONVERT TO INTEGERS P1,P2.
C
14  CALL RDIV(B,R,T)
      CALL RDIV(A,R,C)
      CALL GTOP(C,P1)
      CALL GTOP(T,P2)
C
C  ADD P2 TO P1 AFTER ADJUSTING SIGNS TO  GET A POSITIVE RESULT.
C
      IP1 = IABS(P1(1))
      IP2 = IABS(P2(1))
      IF(P1(1)*P2(1).GT.0) GO TO 70
      IF(IP1-IP2) 30,20,40
20  IF(P1(IP1).GT.P2(IP2)) GO TO 40
30  IF(P1(1).LT.0) GO TO 80
35  P1(1) = -P1(1)
      P2(1) = -P2(1)
      CNEG = -1
      GO TO 80
40  IF(P2(1).LT.0) GO TO 80
      GO TO 35
70  IF(P1(1).LT.0) GO TO 35
80  II = MAX0(IP1,IP2)
      PT(2)=0
      DO 90 I=2,II
      PT(I+1)=0
      PT(I) = P1(1)/IP1*P1(I)*PF(I,IP1)+PF(I,IP2)*P2(1)/IP2*P2(I)
1  +PT(I)
      IF (PT(I).GE.0) GO TO 85
      PT(I) = PT(I) + PBASE
      PT(I+1) = PT(I+1) - 1
      GO TO 90
85  IF(PT(I).LT.PBASE) GO TO 90
      PT(I) = PT(I) - PBASE
      PT(I+1) = PT(I+1) + 1
90  CONTINUE
      IF(PT(II+1).NE.0) II = II + 1
      DO 100 I=2,II
100  P1(I) = PT(I)
      P1(1) = II * CNEG
C
C  CONVERT FINAL NUMERATOR TO INDEXED PRIME FACTORIZATION FORM,
C  DIVIDE BY R, AND PLACE ANSWER IN C.
C
      CALL PTGC(P1,T)
      B(1,1) = B(1,1)*BNEG
      CALL RMPY(T,R,C)
      RETURN
110  WRITE(6,1000)
1000  FORMAT(5X,'ADDITION OPERAND HAS MORE THAN ONE UNFACTORED '
1  , 'COMPONENT; RADD WILL YIELD INCORRECT ANSWER.')
      RETURN
      END

```



```

SUBROUTINE WRITE(A)
IMPLICIT INTEGER*2(A-H,J-O,0-Z)
IMPLICIT INTEGER*4(I,P)
INTEGER*4 MAX0,MIN0
COMMON /COM/ PLIM,PBASE,PT(100),P(10000)
COMMON /RT/R(2,50),T(2,50)
DATA IDASH/'----'/
COMMON /P12/P1(50),P2(50)
DIMENSION A(2,1),IFM(30),IFORM(5),IF1(12),IF2(12),IF3(5)
EQUIVALENCE (IFM(1),IF1(1)),(IFM(13),IF2(1)),(IFM(25),IF3(1))
DATA IF1/'(1H+,23X,27I4)      (1H+,39X,22I4,7X,30I4) '/
DATA IF2/'(1H+,55X,19I4,7X,30I4) (1H+,71X,15I4,7X,30I4) '/
DATA IF3/'(1H+,87X,10I4,7X,30I4) '/

```

C*****

```

C
C
C THIS ROUTINE WRITES OUT TWO INTEGERS
C TO REPRESENT A BASE 10 RATIONAL NUMBER.
C
C

```

C*****

```

      II = A(1,1)
      II = IABS(II)
      IF(II.LE.1) GO TO 26
      IA=2
      IR=2
      IT=2

```

```

C
C PLACE THE NUMERATOR (POSITIVE EXPONENTS) IN T; DENOMINATOR
C (NEGATIVE EXPONENTS) IN R.
C

```

```

      R(2,1)=1
      T(2,1)=1

```

```

C
C CONVERT BOTH TO INTEGERS AND WRITE.
C

```

```

      2 IF(A(1,IA))4,4,6
      4 R(2,IR)=A(2,IA)
        R(1,IR)=-A(1,IA)
        IR=IR+1
        GO TO 7
      6 T(2,IT)=A(2,IA)
        T(1,IT)=A(1,IA)
        IT=IT+1
      7 IA=IA+1
        IF(IA.LE.II)GO TO 2
        R(1,1)=IR-1
        R(2,1)=R(2,IR-1)
        T(1,1) = IT-1
        T(2,1)=T(2,IT-1)
      25 II = A(1,1)/II
        IR=IR-1
        IT=IT-1
        GO TO 29
      26 T(1,1)=A(1,1)
        T(2,1)=1
        R(1,1)=1
        R(2,1)=1
      29 CALL GTOP(T,P1)
        IS=0
      30 IF(T(2,IT-1).LE.8000.OR.IT.EQ.2)GO TO 33
        IT=IT-1
        IS=IS+1
        ISS=-1
        PT(IS)=99999
      31 IS=IS+1
        ISS=ISS+1
        PT(IS)=IABS(P(T(2,IT)+ISS))
        IF(P(T(2,IT)+ISS+1).LT.0.AND.P(T(2,IT)+ISS+1).GT.
      1 -9999999)GO TO 31
        GO TO 30
      33 IS=IS+1

```



```

PT(IS)=99999
I1=IABS(P1(1))-1
IFR= MOD(I1,20)/4
DO 99 IG=1,6
99 IFORM(IG)=IFM(IG+6*IFR)
WRITE(6,1001) (P1(I1+2-I),I=1,I1)
IF(IS.GT.1)WRITE(6,IFORM)(PT(IS+2-I),I=2,IS)
IS=0
CALL GTGP(R,P2)
34 IF(R(2,IR-1).LE.8000.OR.IT.EQ.2)GO TO 37
IR=IR-1
IS=IS+1
ISS=-1
PT(IS)=99999
35 IS=IS+1
ISS=ISS+1
PT(IS)=P(R(2,IR)+ISS)
IF(P(R(2,IR)+ISS+1).LT.0.AND.P(R(2,IR)+ISS+1).GT.
1 -9999999)GO TO 35
GO TO 34
37 IS=IS+1
PT(IS)=99999
I2=IABS(P2(1))-1
IJ=MIN0(MAX0(I1,I2),20)
WRITE(6,1002) I1,(IDASH,IK=1,IJ)
IFR= MOD(I2,20)/4
DO 98 IG=1,6
98 IFORM(IG)=IFM(IG+6*IFR)
WRITE(6,1001)(P2(I2+2-I),I=1,I2)
IF(IS.GT.1) WRITE(6,IFORM) (PT(IS+2-I),I=2,IS)
RETURN
1001 FORMAT(5(6X,5(1X,20I4)))
1002 FORMAT(2X,I2,' * ',20A4)
END

```

```

SUBROUTINE CREATE(I,A)
IMPLICIT INTEGER*2(A-H,J-O,Z)
IMPLICIT INTEGER*4(I,P)
COMMON /P12/PI(50),P2(50)
DIMENSION A(1,1)
COMMON /COM/ PLIM,PBASE,PT(10100)

```

```

C
C*****
C
C
C THIS ROUTINE CREATES THE INDEXED PRIME REPRESENTATION FORM OF
C ANY INTEGER I REPRESENTABLE BY THE MACHINE. PI IS REDUCED TO 1.
C
C
C*****
C
IM = 2
II = IABS(I)
IF(II.LE.1) GO TO 20
PI(1) = I/II
1 PI(IM) = MOD(II,PBASE)
II = (II-PI(IM))/PBASE
IF(II.EQ.0)GO TO 10
IM = IM+1
GO TO 1
10 PI(1) = PI(1)*IM
CALL PTGG(PI,A)
RETURN
20 PI(1) = 2
PI(2) = II
IF(II.NE.0) PI(1) = PI(1)*I
A(1,1) = I
A(2,1)=1
RETURN
END

```



```

SUBROUTINE ENTER(A)
IMPLICIT INTEGER*2(A-H,J-O,Q-Z)
IMPLICIT INTEGER*4(I,P)
COMMON /COM/ PLIM,PBASE,PT(100),PR(10000)
COMMON /P12/ P(50),P2(50)
C
C*****
C
C THIS ROUTINE READS LONG INTEGERS FROM CARDS AND CONVERTS THEM
C TO A. PI IS SET TO 1. THE FORMAT IS
C FIRST CARD:
C   IN=GREATEST INTEGER PART OF (#OF DIGITS)/4 IN COL. 1-5;
C   SIGN = <-1,+1> IN COLUMNS 6,7.
C FOLLOWING CARDS:
C   DIGITS OF THE INTEGER IN GROUPS OF FOUR DIGITS, RIGHT JUSTIFIED
C   WITHIN GROUPS, GROUPS LEFT JUSTIFIED.
C EXAMPLE:  -177133
C   CARD ONE #00002-1#
C   CARD TWO #001771330000...0000#
C
C*****
      READ(5,1000) IN,ISIGN
      READ(5,1001) (P(IN+2-I),I=1,IN)
      P(1) = (IN+1)*ISIGN
      CALL PTGG(P,A)
1000 FORMAT(I5,I2)
1001 FORMAT(5(20I4))
      RETURN
      END

```


U. S. ATOMIC ENERGY COMMISSION
UNIVERSITY-TYPE CONTRACTOR'S RECOMMENDATION FOR
DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

1. AEC REPORT NO.

C00-1469-0223

2. TITLE

RATION--A Rational Arithmetic Package

3. TYPE OF DOCUMENT (Check one):

☒ a. Scientific and technical report

☐ b. Conference paper not to be published in a journal:

Title of conference _____

Date of conference _____

Exact location of conference _____

Sponsoring organization _____

☐ c. Other (Specify) _____

4. RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

☒ a. AEC's normal announcement and distribution procedures may be followed.

☐ b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.

☐ c. Make no announcement or distribution.

5. REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY: NAME AND POSITION (Please print or type)

C. W. Gear

Professor and Principal Investigator

Organization

Department of Computer Science

University of Illinois at Urbana-Champaign

Urbana, Illinois 61801

Signature

Charles W. Gear

Date

June 1973

FOR AEC USE ONLY

7. AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

8. PATENT CLEARANCE:

☐ a. AEC patent clearance has been granted by responsible AEC patent group.

☐ b. Report has been sent to responsible AEC patent group for clearance.

☐ c. Patent clearance not required.

L E it	GRAPHIC DATA	1. Report No. UIUCDCS-R-73-572	2.	3. Recipient's Accession No.
	Title and Subtitle RATION--A Rational Arithmetic Package			5. Report Date June 1973
utor(s)	R. L. Brown			6.
performing	Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801			8. Performing Organization Rept. No. UIUCDCS-R-73-572
ip	sponsoring Organization Name and Address US AEC Chicago Operations Office 9800 South Cass Avenue Argonne, Illinois 60439			10. Project/Task/Work Unit No.
plementary	Notes			11. Contract/Grant No. US AEC AT(11-1)1469
bstracts	<p>In section A, the theoretical basis for representing a rational number using the prime factorization of its minimal integer ratio is given, and the resulting algorithms for performing addition, subtraction, multiplication, and division are developed. The conjecture that these algorithms will normally require fewer machine operations than algorithms based on storing rational numbers as two long integers is supported by a worst case machine operation count estimate for each. These operation counts are applied to parameters derived from test results.</p> <p>Section B is a self-contained user's manual for RATION, a rational arithmetic package written in FORTRAN and based on the above algorithms.</p>			
ey	Words and Document Analysis. 17a. Descriptors rational arithmetic			
de	ifiers/Open-Ended Terms			
TI	Field/Group			
ail	Availability Statement unlimited distribution			
IT	35 (10-70)		19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 41
			20. Security Class (This Page) UNCLASSIFIED	22. Price



UNIVERSITY OF ILLINOIS-URBANA



3 0112 004455322